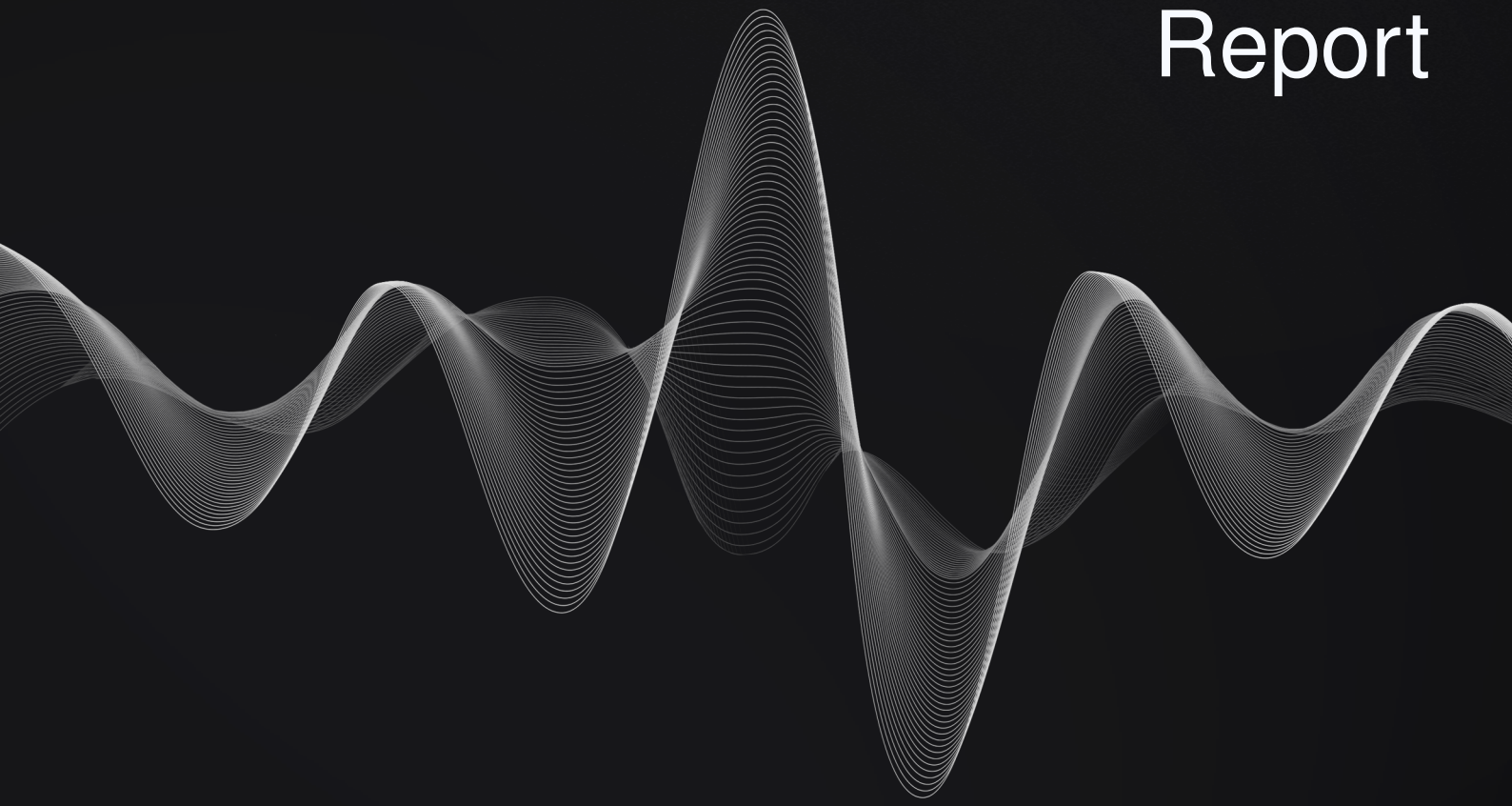
RESONANCE

Meta Pool

MetaPool

Liquid Staking NEAR Smart Contract Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.1)

Audit_Report_MTAP-STN_FINAL_21

Jul 2, 2025	v0.1	João Simões: Initial draft
Jul 11, 2025	v0.2	João Simões: Added findings
Jul 14, 2025	v1.0	Charles Dray: Approved
Jul 29, 2025	v1.1	João Simões: Reviewed findings
Aug 8, 2025	v2.0	Charles Dray: Finalized
Aug 8, 2025	v2.1	Charles Dray: Published

Points of Contact	Pablo	Meta Pool	pablo@metapool.app
	Lucas	Meta Pool	lucas@metapool.app
	Charles Dray	Resonance	charles@resonance.security
Testing Team	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security

Copyright and Disclaimer

Contents

1 Document Control 2

Copyright and Disclaimer 2

2 Executive Summary 4

System Overview 4

Repository Coverage and Quality 4

3 Target 6

4 Methodology 7

Severity Rating 8

Repository Coverage and Quality Rating 9

5 Findings 10

Misleading Account Unregistration Result 11

Missing FtMint And FtBurn Events Emission 12

Pausable Functionality Not Used 13

Usage Of Outdated staked_amount Due To Rounding 14

Unused Functions 15

Dead Code 16

Usage Of Outdated Packages 17

Missing Usage Of NEAR SDK Integer Types For Input And Output 18

Redundant Code Throughout The Protocol 19

Storage-Mutating Function Marked As Read-Only 20

Usage Of std Vec Not Gas Efficient 21

A Proof of Concepts 22

Executive Summary

MetaPool contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between June 30, 2023 and July 14, 2023. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 10 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

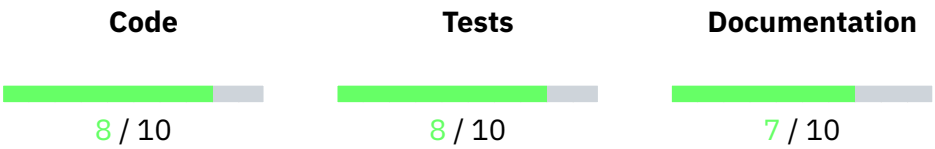
The ultimate goal of the audit was to provide MetaPool with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

Meta Pool is a multi-chain liquid staking ecosystem that enables users to earn yield and participate in governance across several major blockchains, including NEAR, Ethereum, Solana, Aurora, ICP, and more. At its core, it offers liquid staking tokens—such as stNEAR and equivalents on other chains—allowing users to stake assets while retaining liquidity and earning rewards. The platform also features Vote-to-Earn incentives, rewarding users for participating in DAO governance, and provides access to liquidity pools and leveraged staking strategies like the Solana Stake Aggregator, supporting assets such as mpSOL, jitoSOL, bSOL, and SOL.

With a user-friendly interface and continuous cross-chain expansion, Meta Pool empowers users to manage staking, liquidity provision, and governance participation in a seamless and decentralized way. Its smart contracts support key features like staking pool diversification and delayed unstaking, offering both flexibility and performance across ecosystems.

Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.

- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is good**.
- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: Meta-Pool/liquid-staking-contract
- Hash: 72102d36a876ec6ee258bef0e75e531f1648eafc

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust NEAR

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust NEAR audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Race conditions caused by asynchronous cross-contract calls
- Frontrunning attacks
- Storage staking
- Potentially problematic storage layout patterns
- Manual state rollbacks in callbacks
- Access control issues

- Denial of service
- Inaccurate business logic implementations
- Unoptimized Gas usage
- Arithmetic issues
- Client code interfacing

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

Impact \ Likelihood	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Misleading Account Unregistration Result		Resolved
RES-02	Missing FtMint And FtBurn Events Emission		Resolved
RES-03	Pausable Functionality Not Used		Resolved
RES-04	Usage Of Outdated staked_amount Due To Rounding		Resolved
RES-05	Unused Functions		Resolved
RES-06	Dead Code		Resolved
RES-07	Usage Of Outdated Packages		Resolved
RES-08	Missing Usage Of NEAR SDK Integer Types For Input And Output		Acknowledged
RES-09	Redundant Code Throughout The Protocol		Resolved
RES-10	Storage-Mutating Function Marked As Read-Only		Resolved
RES-11	Usage Of std Vec Not Gas Efficient		Acknowledged

Misleading Account Unregistration Result

Low | **RES-MTAP-STN01** | Business Logic | **Resolved**

Code Section

- metapool/src/empty_nep_145.rs#L56-L72

Description

The function `storage_unregister()` returns misleading information whenever the user account that calls the function does not exist, i.e. returns `true` when no account was unregistered. This function should return `false`, or an error indicating that no such account was found on storage.

Misleading information may lead to malformed decisions and subsequent actions by users, which may ultimately damage the trust in the protocol.

Recommendation

It is recommended to validate that the calling user's account exists when being unregistered. Otherwise, an error should be returned.

Status

The issue has been fixed in 620abf5b8aedef3405c678564f305048312bf9c1.

Missing FtMint And FtBurn Events Emission

Low | RES-MTAP-STN02 | Business Logic | Resolved

Code Section

- metapool/src/lib.rs#L419-L425
- metapool/src/lib.rs#L440-L449
- metapool/src/lib.rs#L454-L463

Description

The events `FtMint` and `FtBurn` are not being emitted on all minting and burning actions. These events are missing from the functions `stake_for_lockup()`, `unstake_all()`, and `unstake_from_lockup_shares()`.

Emitted events are vital for transparency, efficiency, and usability in blockchain ecosystems, and, when available, should be used at all times as they enable off-chain users and applications to monitor, index, and respond to on-chain activity.

Recommendation

It is recommended to properly emit these events where applicable and necessary, as to effectively interact with off-chain users and components.

Status

The issue has been fixed in d67551565fa715d84e49d152b7c69cf7ca56e223.

Pausable Functionality Not Used

Low | **RES-MTAP-STN03** | Business Logic | **Resolved**

Code Section

- metapool/src/owner.rs#L20-L30

Description

The smart contracts implement a Pausable functionality, however, it is not being used. Such functionality may be useful during emergencies and should be controlled by multi-signature wallets or decentralized autonomous organizations (DAO).

Recommendation

It is recommended to make use of the Pausable functionality should any emergencies occur within the protocol, in order to protect users, the protocol, and the funds.

Status

The issue has been fixed in cb3e9cd5cdb91b40a1954f8c0b52d9ffb71f9b98.

Usage Of Outdated staked_amount Due To Rounding

Low | RES-MTAP-STN04 | Business Logic | Resolved

Code Section

- metapool/src/lib.rs#L389-L393
- metapool/src/internal.rs#L151

Description

When attempting to deposit and stake NEAR, the function `deposit_and_stake()` is called. This function internally calls `internal_deposit()` and then `internal_stake_from_account()`. The function `internal_stake_from_account()` stakes the available account deposit by `amount`, which may or may not be the same `amount` specified by the user (due to rounding on `is_close()`). This same `amount` is then used on `nslp_try_internal_clearing()`, which may be different from what was actually staked, by a maximum range difference of 0.001 NEAR.

Recommendation

It is recommended to either re-read the actual staked amount or return the staked amount from the function `internal_stake_from_account()` to be used as an input parameter for the function `nslp_try_internal_clearing()`.

Status

The issue has been fixed in c0f358eb096a3d2098c6f51932bf4e3872ae074c.

Unused Functions

Info | **RES-MTAP-STN05** | Code Quality | **Resolved**

Code Section

- metapool/src/reward_meter.rs#L39-L42
- metapool/src/utils.rs#L24-L31
- metapool/src/utils.rs#L75-L77

Description

The following functions and modifiers were found to be unused within the system:

- `RewardMeter::reset()`
- `assert_min_balance()`
- `apply_multiplier()`

Unused functions increase the complexity and readability of the smart contract's code and their inclusion should be discouraged whenever possible.

Recommendation

It is recommended to remove unused functionalities from production-ready code.

Status

The issue has been fixed in 0dec5b31f2ec099eb79cca7d62f34e30878ea3f3.

Dead Code

Info | **RES-MTAP-STN06** | Code Quality | **Resolved**

Code Section

- metapool/src/lib.rs#L350
- metapool/src/lib.rs#L354-L358
- metapool/src/lib.rs#L406-L408
- metapool/src/lib.rs#L413-L415
- metapool/src/lib.rs#L536-L538
- metapool/src/lib.rs#L884-L887
- metapool/src/lib.rs#L894-L896

Description

Throughout the source code there are multiple instances that lead to a dead end as unimplemented features and functionalities are explicitly reverted. These should be cleaned up to increase the quality of the code and decrease the bytecode stored on the blockchain.

It should be noted that while some code might be used to maintain interoperability between other contracts on the blockchain, it may still increase smart contract size while not being entirely necessary, since the blockchain may revert nonetheless.

Recommendation

It is recommended to clean up the code by removing unnecessary logic that increases the protocol complexity and decreases code readability, while decreasing the size of the code on the blockchain.

Status

The issue has been fixed in 461c8943262d58a4dac98907d2ef59b1c7372fda.

Usage Of Outdated Packages

Info	RES-MTAP-STN07	Code Quality	Resolved

Code Section

- Not specified.

Description

The following Rust crates are used as dependencies of the project and contain known vulnerabilities:

- `paste` (1.0.15). `paste` is unmaintained. For more information: RUSTSEC-2024-0436
- `wee_alloc` (0.4.5). `wee_alloc` is unmaintained. For more information: RUSTSEC-2022-0054
- `borsh` (0.7.2). Parsing borsh messages with ZST which are not-copy/clone is unsound. For more information: RUSTSEC-2023-0033
- `borsh` (0.8.2). Parsing borsh messages with ZST which are not-copy/clone is unsound. For more information: RUSTSEC-2023-0033

It should also be noted that the NEAR SDK version `3.1.0` is also outdated and should be bumped to more recent versions to include the latest logical, performance, and security fixes.

Recommendation

It is recommended to use more recent version of the identified crates that solve the identified security vulnerabilities, or otherwise stop using the dependency altogether.

Status

The issue has been fixed in 8d0624bcb2fee8cce8876a26dc154c5a4e1277c3.

Missing Usage Of NEAR SDK Integer Types For Input And Output

Info | **RES-MTAP-STN08** | Code Quality | **Acknowledged**

Code Section

- metapool/src/lib.rs#L560

Description

The function `get_number_of_accounts()` makes use of intput parameters of type `u64`. This type is longer than 52 bits and, as such, is not serializable by JSON, therefore making it impossible to retrieve a properly decoded value when calling this function through an external integrated interface.

Recommendation

It is recommended to make use of NEAR SDK capitalized types in favor of the `i64-i128/u64-u128` native types when dealing with input and output parameters:

- `u64` - `U64`
- `u128` - `U128`
- `i64` - `I64`
- `i128` - `I128`

Status

The issue was acknowledged by MetaPool's team. The development team stated "The use of u64 is intentional, serde-json codifies that as a javascript Number (IEEE 754) and as such it is compatible with other interfaces. It is correct that "Number" has 52 bits of precision and so serde-json can not properly serialize an u64 if it is > 2^53 + 1, but, this being used for "number of accounts" we're comfortable using u64 in this case instead of the U64 (string) version.".

Redundant Code Throughout The Protocol

Info | **RES-MTAP-STN09** | Code Quality | **Resolved**

Code Section

- metapool/src/lib.rs#L366-L371
- metapool/src/lib.rs#L376-L381
- metapool/src/lib.rs#L651-L658
- metapool/src/internal.rs#L454-L474

Description

It was observed that throughout the protocol there are multiple instances of redundant code on several accounts:

- Variables and values related to testing environments;
- Deprecated variables and values;
- Reimplemented standard trait implementations;
- Redundant functions, e.g. `withdraw_all()` and `withdraw_unstaked()`;

These design patterns increase code complexity and do not maximize transaction gas and storage efficiency on the blockchain.

Recommendation

It is recommended to revise code reusability development patterns throughout the protocol, not only to improve readability, but also to maximize gas and storage efficiency on the blockchain. For the specific case of invariant testing, the usage of the function `assert!()` is recommended to differentiate coding patterns of both invariant and valid variable conditions checking, otherwise, NEAR's `require!()` should be used.

Status

The issue has been fixed in 8d0624bcb2fee8cce8876a26dc154c5a4e1277c3.

Storage-Mutating Function Marked As Read-Only

Info | **RES-MTAP-STN10** | Code Quality | **Resolved**

Code Section

- metapool/src/fungible_token_standard.rs#L181-L185

Description

The function `ft_metadata_set()` is marked as read-only, however, it mutates the storage of the smart contract, specifically the `LazyOption` with storage key `ftmd`. This fact may create inconsistencies in the usage of the protocol by its users.

Recommendation

It is recommended to properly mark te function `ft_metadata_set()` as a call function and ensure that the relevant collection is part of the contract's state and storage.

Status

The issue has been fixed in a8924385c2d71575a9f407821175ab4248a2e889.

Usage Of std Vec Not Gas Efficient

Info	RES-MTAP-STN11	Gas Optimization	Acknowledged

Code Section

- metapool/src/lib.rs#L194

Description

The state variable `staking_pools` makes use of `Vec` as a collection from Rust's `std` library. This collection, as opposed to NEAR SDK's `Vector` collection, is not as efficient in term of gas usage when dealing with large amounts of data that do not need to be accessed altogether.

Recommendation

It is recommended to switch to the more gas efficient NEAR SDK `Vector` collection for the variable `staking_pools`.

Status

The issue was acknowledged by MetaPool's team. The development team stated "Using a Vec instead of a Vector is a deliberate choice. The idea is to manage the list as a whole to keep the invariant sum(weights)=100%. We've run tests in the previous audit and the system works properly for up to a Vec with 3400 pools. Considering the low price of Gas in NEAR, we deliberately selected this option.".

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.